

EFFICIENT LLM INFERENCE WITH KCACHE

Qiaozhi He, Zhihua Wu
qiaozhihe2022@outlook.com

ABSTRACT

Large Language Models (LLMs) have had a profound impact on AI applications, particularly in the domains of long-text comprehension and generation. KV Cache (Pope et al., 2022) technology is one of the most widely used techniques in the industry. It ensures efficient sequence generation by caching previously computed KV states. However, it also introduces significant memory overhead. We discovered that KV Cache is not necessary and proposed a novel KCache technique to alleviate the memory bottleneck issue during the LLMs inference process. KCache can be used directly for inference without any training process. Our evaluations show that KCache improves the throughput of popular LLMs by 40% with the baseline, while keeping accuracy.

1 INTRODUCTION

Currently, LLMs like GPT-4 (OpenAI, 2023), PaLM (Chowdhery et al., 2022; Anil et al., 2023), LLaMA 3 (AI@Meta, 2024) dominate in numerous natural language processing, summary, code generation, question answering, etc. However, their expensive online inference cost poses significant obstacles to the deployment of LLM-based applications. With limited computational resources, how to maximize the overall system throughput as much as possible, and improving the utilization rate of the GPU cluster becomes increasingly important. LLMs inference consists of two phases: *prefill* phase and *decode* phase. The *decode* phase generates tokens one by one, based on the result of *prefill* phase and the previous step of *decode* phase, which is memory bandwidth bound. So, we need to increase the batch size to improve the system throughput, but increasing the batch size will further occupy more GPU memory.

The memory usage of LLM inference mainly consists of 3 parts: model weights, activations, and KV Cache. For Instance, For the LLaMA2-7B model, the weights occupy around 14GB of memory at fp16 precision. When processing a batch size of 8 and a sequence length of 32×1024 , the KV cache occupies around 128GB of memory, with the layer-wise memory sharing strategy, the activations only occupy 2GB of memory. KV Cache = $2 \times \text{bytes} \times \text{bsdl}$, where 2 represents K Cache and V Cache, b represents the batch size, s represents the sequence length and d represents the embedding dimension and l represents the number of layers. As the batch size and sequence length increase, the memory usage of the KV Cache will increase linearly.

Some optimizations have been proposed to alleviate the KV Cache bottleneck. Quantization compression algorithms, (Dong et al., 2024; Kang et al., 2024; Yue et al., 2024) have been proposed to compress the KV Cache from the *bytes* perspective. Context window compression algorithms, (Zhang et al., 2023; Xiao et al., 2024; Liu et al., 2023) have been proposed to compress the KV Cache from s perspective. Adaptive computation algorithms, (Schuster et al., 2022) early exit decoding to reduce compute, which from l perspective. (Shazeer, 2019; Ainslie et al., 2023) accelerates inference by improving the structure of the Multi-Head Attention (MHA). From the K Cache and V Cache perspective, although simply offloading to CPU and reloading back to GPU during inference can alleviate the pressure on GPU memory, the current Host-to-Device (H2D) and Device-to-Host (D2H) bandwidth will become the new bottleneck for inference.

(Zhang et al., 2023; Xiao et al., 2024; Liu et al., 2023) have been proposed that only pivotal tokens are important during inference, which KV Cache is compressed by deleting part of them. However, considering multi-turn question-answering scenarios, deleting parts of the KV Cache directly without a fallback mechanism is a highly risky action. A more flexible approach is to retain all KV states as much as possible and dynamically select the key information for computation. This way, since all KV states are preserved, the upper bound of accuracy can be guaranteed to be high enough. Based on this idea, an obvious method is to offload all KV states to CPU memory. Another key issue is how to dynamically select which KV states are important and copy them back to HBM from CPU memory for attention calculation. As long as this partial information can maximally preserve all semantic information, the inference accuracy can approach the theoretical upper bound as much as possible, while the partial data copying can also maximize the inference performance. We propose **KCache**. During the inference process, we retain the K Cache in HBM while storing the V Cache in CPU Memory. Simultaneously, we directly utilize the softmax results from the Attention computation to filter out the key

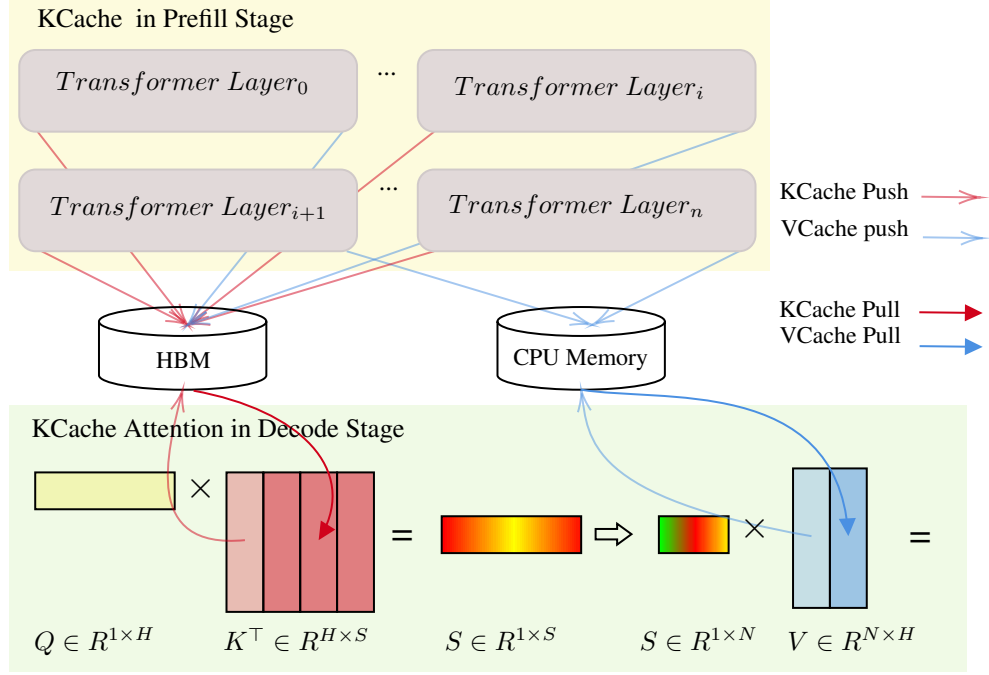


Figure 1: **Illustration of the KCache.** During *prefill* phase, the computation results of each layer *push* to the HBM. After that, the part of V Cache will be copied to the CPU asynchronously, while releasing the GPU memory occupied by this part of the V Cache. During *decode* phase, K states will be pushed and pulled as KV Cache. However, we will calculate the *topN* of attention scores, and based on the indices of the topN results, we will pull the corresponding V Cache from the CPU to the HBM in real-time to complete the subsequent computation.

information and recall the corresponding V Cache from CPU Memory for subsequent Attention calculations. Through this simple approach, leveraging the structural characteristics of Transformer models, we effectively utilize the idle CPU memory, increasing the capacity of HBM.

In this paper, we build *InferenceEngine* based on KCache that efficiently reduces the memory footprint during LLM inference, which achieved 40% increased throughput and keeping accuracy. The main contributions of our work include:

- We propose KCache that can be used directly for inference without any training process while improving throughput by 40% while maintaining accuracy.
- We identified the performance and accuracy challenges in offloading the VCache to CPU memory, proposed KCache to address this challenge, and validated its effectiveness through experiments on model inference.
- KCache is flexible and scalable, which can be applied to transformed pre-trained models.

2 BACKGROUND

In this section, we present some basic knowledge of LLMs, which include *autoregressive inference*, *prefill* and *decode*.

LLMs are essentially based on a decoder-only architecture, which consists of L stacked blocks. Each block includes two modules: a multi-head attention (MHA) (Vaswani et al., 2023) and a fully connected feed-forward network (FFN). An input tensor $x \in R^{b \times s \times d}$, where b represents the batch size, s represents the sequence length of input, and d represents the hidden dimension of the model. MHA maps the input x to different subspaces using n heads: $H^i = \text{softmax}(Q^i(K^i)^T / \sqrt{d_h})V^i$, $MHA(x) = \text{Concat}(H^1, H^2, \dots, H^{n-1}, H^n)W_o$, where $Q^i = xW_{q_i}$, $K^i = xW_{k_i}$, $V^i = xW_{v_i}$, and $W_{q_i} \in R^{h \times h}$, $W_{k_i} \in R^{h \times h}$, $W_{v_i} \in R^{h \times h}$ are trainable weights, h represents the hidden dimension of per head. $d_h = d/n$. FFN take SwiGLU (Shazeer, 2020) for example, $FFN_{SwiGLU}(x, W, V, W_2) = (\sigma(xW) \otimes xV)W_2$, where $W \in R^{d \times 8/3d}$, $V \in R^{d \times 8/3d}$, $W_2 \in R^{8/3d \times d}$, σ is unit of activation. LLMs process a sequence of words named *prompt* and generate some new words. The

autoregressive inference means that the token generated at the current moment depends on the token generated at the previous moment. The process of handling user prompts is called *prefill*, and it only needs to be done once. The process of generating all output tokens one by one in autoregression is called *decode* and needs to be executed continuously. During the *prefill* phase, taking prompt as input and computation in parallel using matrix-matrix multiplications. During the *decode* phase, which performs the same operations as *prefill*, but only takes one token as input and computation with KV Cache using vector-matrix multiplications.

3 METHOD

3.1 KCACHE

	Submodule	FLOPs	I/O(byte)	Arithmetic Intensity
MHA	$Q = xW_q, K = xW_k, V = xW_v$	$6bd^2$	$12bd + 6d^2$	$\frac{1}{\frac{2}{d} + \frac{1}{b}} \approx b$
	$S = \text{softmax}(QK^\top / \sqrt{d_h})$	$2bsd$	$2bnh + 2bnhs + 2bns$	$\frac{1}{1 + \frac{1}{h} + \frac{1}{s}} \approx 1$
	$A = SV$	$2bsd$	$2bns + 2bsd + 2bd$	$\frac{1}{1 + \frac{1}{h} + \frac{1}{s}} \approx 1$
	$O = AW_o$	$2bd^2$	$4bd + 2d^2$	$\frac{1}{\frac{2}{d} + \frac{1}{b}} \approx b$
KCache MHA	$Q = xW_q, K = xW_k, V = xW_v$	$6bd^2$	$12bd + 6d^2$	$\frac{1}{\frac{2}{d} + \frac{1}{b}} \approx b$
	$\tilde{S} = \text{TopN}(\text{softmax}(QK^\top / \sqrt{d_h}))$	$2bsd$	$2bnh + 2bnhs + 2bnN$	$\frac{1}{1 + \frac{N}{sh} + \frac{1}{s}} \approx 1$
	$\tilde{A} = \tilde{S} \text{Part}(V)$	$2bNd$	$2bnN + 2bNd + 2bd$	$\frac{1}{1 + \frac{1}{h} + \frac{1}{s}} \approx 1$
	$\tilde{O} = \tilde{A}W_o$	$2bd^2$	$4bd + 2d^2$	$\frac{1}{\frac{2}{d} + \frac{1}{b}} \approx b$

Table 1: **MHA FLOPs and I/O(byte) in *decode* phase.** N denotes that the value of N selected for the TopN operation.

In long-context scenarios, users typically ask multiple rounds of questions based on a long sequence, with each question potentially focusing on different segments of the long context. To maximize the accuracy of results in each round, we avoid reducing or compressing the KV states, thus ensuring the upper bound of model effectiveness. However, simply offloading KV states to CPU memory and reloading them to the GPU during inference would significantly increase the end-to-end inference time. Therefore, to balance model effectiveness and inference latency, we must find a way to reload only the necessary information back to HBM, which implies the need for a module to determine which information is important. Fortunately, considering the meaning of the Key and Value pairs in the Attention mechanism, where Key is used to compute the relevance with Query and Value represents the actual information associated with Key, it inspires us to offload a portion of the K Cache and V Cache to CPU memory.

Figure 1 shows the method of KCache. We keep K Cache and first of $0 \dots i$ layers V Cache in HBM and keep other V Cache in CPU memory. During computation, The attention computation is adjusted from $\text{softmax}(QK^\top / \sqrt{d_h})$ to $\text{TopN}(\text{softmax}(QK^\top / \sqrt{d_h}))$. Since the K Cache is still stored in HBM, the computation of $QK^\top$ is not affected. After the softmax operation, TopN selects the N most relevant results. We dynamically and flexibly move the corresponding vectors of the V Cache to HBM in real-time based on the attention scores, to participate in subsequent calculations.

Based on the proposed KCache method, intuitively, as N increases, the model’s inference accuracy will approach that of the full KV Cache, but it will also increase the data copying overhead, leading to performance degradation. Whether there exists a perfect balance between inference performance and inference accuracy requires quantitative analysis. In the following sections, we provide an analysis from both the accuracy and performance perspectives.

3.2 ANALYSIS OF KCACHE PERFORMANCE

During the *prefill* phase, the part of V Cache needs to be asynchronously copied to the CPU memory. We hope that the computation time for each layer can overlap the data copying time of the previous layer. There are $2bsd$ bytes data needed to transmit from Device to Host for each transformer block, and $22bsd^2 + 4bs^2d$ floating point operations(FLOPs) for each Transformer Block. Let:

$$\frac{22bsd^2 + 4bs^2d}{FLOPS} > \frac{2bsd}{Bandwidth_{D2H}} \quad (1)$$

$$11d + 2s > \frac{FLOPs}{Bandwidth_{D2H}} \quad (2)$$

Take NVIDIA A100 (80GB) GPU for instance, $d = 4096$ for LLaMA2-7B, which means computation will overlap the transmission.

During the *decode* phase, the Multi-Head Attention (MHA) module is a typical memory-bound task, as evidenced by its Arithmetic Intensity shown in Table 1. The Arithmetic Intensity is defined as the ratio of floating-point operations (FLOPs) to I/O bytes. This indicates that the computation time of the MHA module during decoding is strongly dependent on the amount of memory access. Notably, the performance of the MHA module in the *decode* phase is independent of the hidden size and sequence length and is solely influenced by the batch size. This observation leads to an expectation: the computation time and data transfer time of the proposed KCache MHA module can be less than the conventional KV cache MHA implementation. Let:

$$\frac{bnNh}{Bandwidth_{H2D}} < \frac{(2bns + 2bsd + 2bd) - (2bnN + 2bNd + 2bd)}{Bandwidth_{GPU}} \quad (3)$$

$$\frac{s}{N} > \frac{Bandwidth_{GPU}}{Bandwidth_{H2D}} \quad (4)$$

Take NVIDIA A100(80GB) GPU (2039GB/s GPU Memory Bandwidth and 32GB/s H2D Bandwidth) for instance, which means KCache performance will not decrease when $s/N > 64$.

3.3 ANALYSIS OF KCACHE ACCURACY

During the *prefill* phase, the *Value* tensors are asynchronously offloaded to CPU memory, which does not affect the inference accuracy and performance. During the *decode* phase, it is necessary to reduce the amount of data transferred from host to device. Based on $S_{b,i} = \text{softmax}(Q^{b,i}(K^{b,i})^\top / \sqrt{d_h}) \in R^{1 \times s}$, $A_{b,i} = S_{b,i}V_{b,i} \in R^{1 \times d}$, where b represents one instance of batch and i represents one of head. If the result of $S_{b,i}$ is sparse enough, the impact of the corresponding value of $A_{b,i}$ on the final result will be negligible. In 4, The accuracy of KCache will be further verified.

	LLaMA2-7B			LLaMA2-13B			LLaMA3-8B			Mistral-13B		
Config	BBH	GSM	TriQA	BBH	GSM	TriQA	BBH	GSM	TriQA	BBH	GSM	TriQA
	3-shot	8-shot	5-shot	3-shot	8-shot	5-shot	3-shot	8-shot	5-shot	3-shot	8-shot	5-shot
Baseline	39.87	15.09	64.24	47.69	25.78	70.64	62.39	47.61	71.71	56.21	40.33	70.94
$N = 32 \ L = 0$	33.19	5.46	63.57	40.73	10.84	70.67	56.46	46.7	71.67	46.54	34.5	70.84
$N = 32 \ L = 1$	36.94	12.59	64.16	45.97	25.32	70.53	56.38	46.32	71.72	52.08	36.62	70.85
$N = 32 \ L = 2$	37.08	14.25	64.12	46.17	26.31	70.49	56.43	46.17	71.7	52.17	37.38	70.85
$N = 32 \ L = 3$	36.63	14.33	64.14	45.68	25.93	70.49	56.41	45.87	71.69	52.37	36.85	70.82
$N = 64 \ L = 0$	35.75	6.14	62.48	45.4	14.4	69.73	60.99	47.54	71.67	54.19	38.29	70.87
$N = 64 \ L = 1$	38.03	13.42	64.18	46.52	26.84	70.61	61.23	48.67	71.66	54.52	37.15	70.88
$N = 64 \ L = 2$	37.97	15.31	64.18	46.98	27.22	70.61	61.1	47.99	71.68	54.38	38.44	70.88
$N = 64 \ L = 3$	38.07	15.16	64.22	46.89	25.55	70.59	61.1	47.84	71.73	54.72	39.04	70.88
$N = 128 \ L = 0$	37.26	8.49	64.07	45.54	18.12	70.63	62.69	47.84	71.72	56.18	37.83	70.94
$N = 128 \ L = 1$	38.57	14.25	64.24	47.23	25.78	70.66	62.79	48.07	71.7	56	37.91	70.93
$N = 128 \ L = 2$	39	15.01	64.24	47.18	25.17	70.67	62.79	48.98	71.72	55.89	37.98	70.92
$N = 128 \ L = 3$	38.89	15.09	64.25	47.17	25.55	70.67	62.52	48.52	71.72	56.26	38.44	70.92

Table 2: KCache results for LLaMA2-7B, LLaMA2-13B, LLaMA3-8B and Mistral-13B. GSM denotes GSM8K, and TriQA denotes TriviaQA. BBH, GSM8K and TriviaQA are measured in accuracy. N denotes that the value of N selected for the TopN operation. L denotes that the layer of VCache allocated on HBM, which means VCache of $layer_0$ and $layer_1$ was allocated on HBM and other layers on CPU memory when $L = 2$. We provide the score of LLaMA3-8B with KCache where $N = 128$ and $L = 1$ on each subject of BBH in Table 4.

4 EXPERIMENTS

4.1 SETUP

Models and Datasets. All models are based on decoder-only transformers. We evaluate KCache on four open-source LLMs: LLaMA2-7B, LLaMA2-13B (Touvron et al., 2023), LLaMA3-8B (AI@Meta, 2024) and Mistral-7B (Jiang et al., 2023). We choose 3 benchmarks: BBH, GSM8K and TriviaQA. BBH (Suzgun et al., 2022) is a suite of 23 challenging BIG-Bench tasks. GSM8K (Cobbe et al., 2021), a dataset of 8.5k high-quality linguistically diverse grade school math word problems. TriviaQA (Joshi et al., 2017), a reading comprehension dataset containing over 650K question-answer-evidence triples.

4.2 RESULTS

Configuration		KVCache	KCache		
Input Output	Batch Size	Throughput tokens/second	Throughput $N = 32$	Throughput $N = 64$	Throughput $N = 128$
1k/1k	1	55.3	43.7	43.4	42.9
	8	321.0	277.1	256.2	222.2
	16	485.9	441.0	389.7	315.7
4k/1k	1	50.7	41.9	41.7	41.2
	8	212.0	225.9	211.8	188.0
	14	251.2	290.8	267.8	231.4
	23	OOM	349.2	316.2	266.2
7k/1k	1	46.7	40.5	40.1	39.6
	8	158.4	189.8	180.1	162.6
	13	OOM	223.2	209.5	186.1
15k/1k	1	38.3	36.5	36.4	36.0
	3	65.9	76.6	75.8	73.7
	5	OOM	102.5	100.4	95.1

Table 3: KCache throughput on LLaMA2-7B. OOM denotes Out of Memory Error. KCache demonstrated performance advantages when handling contexts longer than 4K, and this advantage further increased as the context length grew. When reaching 15K, KCache exhibited over 40% higher throughput compared to the baseline.

Accuracy. We run all tasks with (Gao et al., 2023) for fair comparison. Table 2 shows experimental results about the Few-shot performance. Fig 2 shows prompt length on different datasets.

- KCache essentially maintained accuracy without loss, and even achieved better performance across multiple datasets and models.
- L has a relatively small impact on the model accuracy, especially when the model performance is sufficiently strong. If the model performance is relatively weak, it is recommended to set a larger L .
- A larger N would achieve higher accuracy. When $N = 128$, KCache maintained the same accuracy as the baseline, or even higher. We believe that TopN regularized the softmax and further filtered out noise information.
- Our experiments on three datasets validated that for context lengths around 2K or less, setting N to either 64 or 128 did not significantly impact the accuracy.

Performance. We further evaluated the end-to-end performance of KCache in our InferenceEngine and conducted experiments on GPU, which has 64GB memory with 1TB GPU memory bandwidth and 180TFLOPS. We evaluate LLaMA2-7B. Table 3 shows the experimental result. Overall, The experimental results further validate the analysis in 3.2, where KCache demonstrates performance advantages when $S \gg N$. Simultaneously,

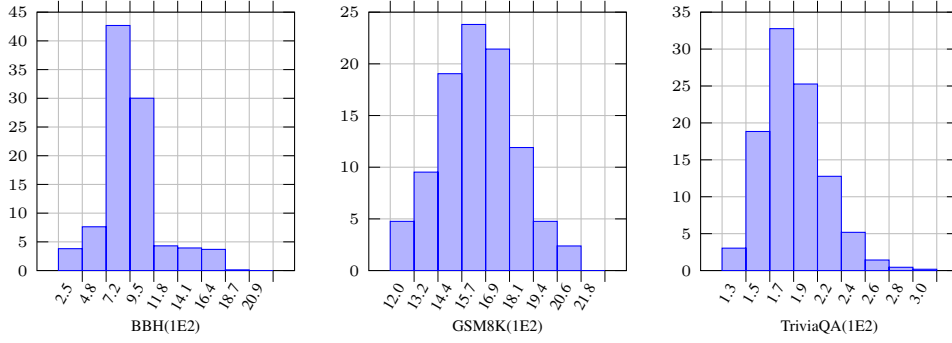


Figure 2: Prompt length of BBH, GSM8K and TriviaQA.

based on the results in 2, KCache achieved a 40%+ throughput improvement in inference with 15K context length with $N = 128$.

5 CONCLUSION

In this work, We propose KCache, an efficient inference technique for large language models. Particularly in long-context inference scenarios, KCache demonstrates a 40%+ throughput improvement. This approach does not require any training and applies to various mainstream structures such as MHA and GQA. In the future, we will further explore strategies based on KCache.

REFERENCES

- AI@Meta. Llama 3 model card. 2024. URL https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md.
- Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints, 2023.
- Rohan Anil, Andrew M. Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, Eric Chu, Jonathan H. Clark, Laurent El Shafey, Yanping Huang, Kathy Meier-Hellstern, Gaurav Mishra, Erica Moreira, Mark Omernick, Kevin Robinson, Sebastian Ruder, Yi Tay, Kefan Xiao, Yuanzhong Xu, Yujing Zhang, Gustavo Hernandez Abrego, Junwhan Ahn, Jacob Austin, Paul Barham, Jan Botha, James Bradbury, Siddhartha Brahma, Kevin Brooks, Michele Catasta, Yong Cheng, Colin Cherry, Christopher A. Choquette-Choo, Aakanksha Chowdhery, Clément Crepy, Shachi Dave, Mostafa Dehghani, Sunipa Dev, Jacob Devlin, Mark Díaz, Nan Du, Ethan Dyer, Vlad Feinberg, Fangxiaoyu Feng, Vlad Fienber, Markus Freitag, Xavier Garcia, Sebastian Gehrmann, Lucas Gonzalez, Guy Gur-Ari, Steven Hand, Hadi Hashemi, Le Hou, Joshua Howland, Andrea Hu, Jeffrey Hui, Jeremy Hurwitz, Michael Isard, Abe Ittycheriah, Matthew Jagielski, Wenhao Jia, Kathleen Kenealy, Maxim Krikun, Sneha Kudugunta, Chang Lan, Katherine Lee, Benjamin Lee, Eric Li, Music Li, Wei Li, YaGuang Li, Jian Li, Hyeontaek Lim, Hanzhao Lin, Zhongtao Liu, Frederick Liu, Marcello Maggioni, Aroma Mahendru, Joshua Maynez, Vedant Misra, Maysam Moussalem, Zachary Nado, John Nham, Eric Ni, Andrew Nystrom, Alicia Parrish, Marie Pellat, Martin Polacek, Alex Polozov, Reiner Pope, Siyuan Qiao, Emily Reif, Bryan Richter, Parker Riley, Alex Castro Ros, Aurko Roy, Brennan Saeta, Rajkumar Samuel, Renee Shelby, Ambrose Slone, Daniel Smilkov, David R. So, Daniel Sohn, Simon Tokumine, Dasha Valter, Vijay Vasudevan, Kiran Vodrahalli, Xuezhi Wang, Pidong Wang, Zirui Wang, Tao Wang, John Wieting, Yuhuai Wu, Kelvin Xu, Yunhan Xu, Linting Xue, Pengcheng Yin, Jiahui Yu, Qiao Zhang, Steven Zheng, Ce Zheng, Weikang Zhou, Denny Zhou, Slav Petrov, and Yonghui Wu. Palm 2 technical report, 2023.
- Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ip-polito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivan Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayanan Pillai, Marie Pellat, Aitor

- Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. Palm: Scaling language modeling with pathways, 2022.
- Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021.
- Shichen Dong, Wen Cheng, Jiayu Qin, and Wei Wang. Qaq: Quality adaptive quantization for llm kv cache, 2024.
- Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac’h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, 12 2023. URL <https://zenodo.org/records/10256836>.
- Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, L  lio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timoth  e Lacroix, and William El Sayed. Mistral 7b, 2023.
- Mandar Joshi, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension, 2017.
- Hao Kang, Qingru Zhang, Souvik Kundu, Geonhwa Jeong, Zaoxing Liu, Tushar Krishna, and Tuo Zhao. Gear: An efficient kv cache compression recipe for near-lossless generative inference of llm, 2024.
- Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time, 2023.
- OpenAI. Gpt-4 technical report, 2023.
- Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Anselm Levskaya, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference, 2022.
- Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Q. Tran, Yi Tay, and Donald Metzler. Confident adaptive language modeling, 2022.
- Noam Shazeer. Fast transformer decoding: One write-head is all you need, 2019.
- Noam Shazeer. Glu variants improve transformer, 2020.
- Mirac Suzgun, Nathan Scales, Nathanael Sch  rli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V. Le, Ed H. Chi, Denny Zhou, and Jason Wei. Challenging big-bench tasks and whether chain-of-thought can solve them, 2022.
- Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023.
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.
- Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks, 2024.

Yuxuan Yue, Zhihang Yuan, Haojie Duanmu, Sifan Zhou, Jianlong Wu, and Liqiang Nie. Wkvquant: Quantizing weight and key/value cache for large language models gains more, 2024.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H₂o: Heavy-hitter oracle for efficient generative inference of large language models, 2023.

A APPENDIX

BBH	KVCache	KCache
bbh_cot_fewshot_boolean_expressions	87.60	88.40
bbh_cot_fewshot_causal_judgement	47.59	54.55
bbh_cot_fewshot_date_understanding	82.40	82.40
bbh_cot_fewshot_disambiguation_qa	46.00	60.80
bbh_cot_fewshot_dyck_languages	10.00	10.80
bbh_cot_fewshot_formal_fallacies	53.60	53.60
bbh_cot_fewshot_geometric_shapes	41.20	41.20
bbh_cot_fewshot_hyperbaton	92.80	92.00
bbh_cot_fewshot_logical_deduction_five_objects	44.40	44.80
bbh_cot_fewshot_logical_deduction_seven_objects	36.40	34.40
bbh_cot_fewshot_logical_deduction_three_objects	79.20	76.40
bbh_cot_fewshot_movie_recommendation	88.40	89.60
bbh_cot_fewshot_multistep_arithmetic_two	31.60	37.20
bbh_cot_fewshot_navigate	88.80	86.80
bbh_cot_fewshot_object_counting	82.80	82.40
bbh_cot_fewshot_penguins_in_a_table	69.18	67.81
bbh_cot_fewshot_reasoning_about_colored_objects	76.40	74.00
bbh_cot_fewshot_ruin_names	69.20	69.60
bbh_cot_fewshot_salient_translation_error_detection	54.00	52.00
bbh_cot_fewshot_snarks	60.11	67.98
bbh_cot_fewshot_sports_understanding	96.00	96.00
bbh_cot_fewshot_temporal_sequences	71.20	66.80
bbh_cot_fewshot_tracking_shuffled_objects_five_objects	36.80	40.40
bbh_cot_fewshot_tracking_shuffled_objects_seven_objects	32.00	28.80
bbh_cot_fewshot_tracking_shuffled_objects_three_objects	61.60	59.60
bbh_cot_fewshot_web_of_lies	100.00	100.00
bbh_cot_fewshot_word_sorting	43.60	38.40

Table 4: The scores of each subject in BBH of LLaMA3-8B.